CodeCV: Mining Expertise of GitHub Users from Coding Activities

Daniel Atzberger, Nico Scordialo, Tim Cech, Willy Scheibel, Matthias Trapp, Jürgen Döllner
Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam, Germany
{daniel.atzberger, tim.cech, willy.scheibel, matthias.trapp, juergen.doellner}@hpi.uni-potsdam.de
nico.scordialo@student.hpi.uni-potsdam.de

***Abstract*—The number of software projects developed collaboratively on social coding platforms is steadily increasing. One of the motivations for developers to participate in open-source software development is to make their development activities easier accessible to potential employers, e.g., in the form of a resume for their interests and skills. However, manual review of source code activities is time-consuming and requires detailed knowledge of the technologies used. Existing approaches are limited to a small subset of actual source code activity and metadata and do not provide explanations for their results.**

In this work, we present *CodeCV*, an approach to analyzing the commit activities of a *GitHub* user concerning the use of programming languages, software libraries, and higher-level concepts, e.g., *Machine Learning* or *Cryptocurrency*. Skills in using software libraries and programming languages are analyzed based on syntactic structures in the source code. Based on Labeled Latent Dirichlet Allocation, an automatically generated corpus of GitHub projects is used to learn the concept-specific vocabulary in identifier names and comments. This enables the capture of expertise on abstract concepts from a user's commit history. *CodeCV* further explains the results through links to the relevant commits in an interactive web dashboard. We tested our system on selected GitHub users who mainly contribute to popular projects to demonstrate that our approach is able to capture developers' expertise effectively.

***Index Terms*—Mining Software Repositories, Developer Expertise, GitHub, Topic Models.**

I. Introduction

With more than 83 million contributors, GitHub is the most popular social coding platform[1]. A significant motivation for participating in the development of open-source software is the creation of a resume that can be used as a reference for technological and social skills in application procedures [1]. A developer's GitHub profile allows a potential employer to get an overview of a developer's interests by looking at his or her own or forked projects, the number of followers, a chart summarizing his or her activities, or badges indicating notable achievements within the open-source community, such as sponsoring open-source contributors[2]. Although GitHub can show a user's involvement in the open-source community, source code activity is the primary source to get detailed information about a user's proficiency in programming languages, software libraries, or higher-level concepts, e.g., *Machine Learning* or *Cryptocurrency*. A. Horowitz cites the following as the main reason for using GitHub over traditional platforms like *LinkedIn* when searching for software engineers: "why would I look at their resume when I can look at a body of work?" suggesting that actual source code is far more meaningful[3]. However, reviewing and assessing the individual commits is time-consuming and requires the recruiter's expertise in each topic area.

Various approaches to analyzing and visualizing developer activity on GitHub were presented [1]–[5]. These only look at small sections of the actual coding activities, e.g., import statements [5] or predefined expressions [4], or rely on metadata provided by GitHub, commit messages, or ReadMe files of the projects a developer has contributed to [1]–[3], and are therefore very limited in their expressiveness. In no case do the approaches provide explanations for their results and therefore cannot assist a potential employer in finding the relevant code sections that indicate expertise.

In this work, we propose a method for processing, analyzing, and visualizing a developer's commit activities to help an employer find expertise in programming languages, software libraries, and higher-level concepts regarding usage expertise [6]. Based on the frequency of library calls, a usage score is calculated. Similarly, the frequency of regular expressions determines the level of experience in using a programming language. By training a Labeled Latent Dirichlet Allocation (LLDA) model, a variant of Latent Dirichlet Allocation (LDA), on an automatically generated corpus, the specific vocabulary for a higher-level concept in comments and identifier names is extracted, allowing the computation of an expertise score for higher-level concepts as presented by Atzberger *et al.* [7]. The corpus consists of GitHub projects together with their associated *GitHub topics*[4] and is generated automatically. Results are displayed in an interactive dashboard with backlinks to commits representative of each concept. We demonstrate the potential of our approach by studying selected contributors to popular projects. In summary, we make the following contributions:

1) **Mining Expertise in higher-level concepts**: Our approach considers the vocabulary in comments and identifier names and does not require predefined rules.

[1]https://github.com/about
[2]https://docs.github.com/en/sponsors/sponsoring-open-source-contributors
[3]https://www.forbes.com/sites/anthonykosner/2012/10/20/software-engineers-are-in-demand-and-github-is-how-you-find-them/?sh=9140df64328d
[4]https://github.com/topics

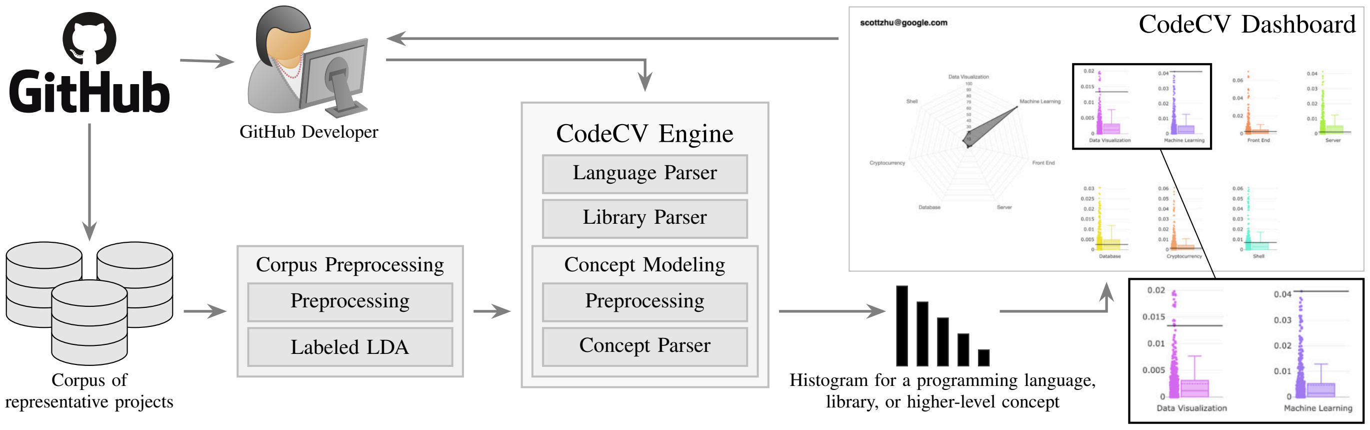


Fig. 1. Data processing pipeline for creating a *CodeCV*. The commit history as well as representative projects are extracted from GitHub. After training an LLDA model, skills in programming languages, libraries and concepts can be parsed. Using a score function on the histogram the results are displayed on the interactive dashboard that allows to trace back to the actual commits.

2) **Explainability**: Our approach allows to create commit-level traceability that helps evaluate a developer's skills.
3) **Implementation**: We have implemented a fully functional prototype[5] and are presenting the results of selected GitHub contributors.

II. Related Work

Analyzing users' activities on social coding platforms and making the corresponding results available to recruiters was the starting point of earlier work. Kuttal *et al*. presented *Visual Resume*, a system for exploring a developer's contributions on GitHub and *Stack Overflow*[6] [1]. By aggregating metadata from their activity traces, e.g., the number of owned or forked projects, or the number of accepted answers, their approach aims to support the hiring process by inferring technological and social skills. The system also visualizes the results and allows direct comparison between two developers. Greene and Fischer proposed an approach to identify expertise in technologies, programming languages, and higher-level concepts for GitHub users [2]. For this purpose, a list of whitelisted terms is extracted from the ReadMe and the commit messages. The developer's knowledge of the programming languages results from the file types of the changed files. The results are visualized in the form of an interactive tag cloud that allows the user to filter for applicants with the required skills. Work similar to ours was presented by Kourtzanidis *et al.* , who developed *RepoSkillMiner*, a system for analyzing and visualizing a GitHub user's familiarity with programming languages and frameworks [4]. Their approach extracts skills for three .NET technologies and two front-end frameworks by applying the *Microsft Language Understanding Intelligent Service* to the actual code chunks modified by the developer. The results are further displayed in a web dashboard using basic two-dimensional charts. Hauff and Gousios developed an algorithm that matches job advertisements with developers on GitHub [3]. To this end, Named Entity Recognition and Named Entity Disambiguation extract concepts, programming languages, and technologies from job descriptions and ReadMe files. After weighting the vectors according to the tf-idf scheme, developers and job descriptions can be compared based on cosine similarity. Another approach, using tags as additional information, was proposed by Saxena and Pedanekar [5]. By annotating import statements in Java files with tags from Stack Overflow, the authors calculated a score expressing expertise in a broad range of topics. In addition to their analysis, the authors visualized a developer's skills profile using a treemap. The presented approaches have the same limitations: Their analyses are based on metadata, which makes a detailed investigation of a developer's contributions to software projects impossible, or they rely on a list of predefined terms meant to indicate expertise in higher-level concepts in the sense of usage expertise [6]. Similar approaches were presented for mining expertise in library usage [8] or higher-level concepts [9]. In all cases, only a tiny part of the actual vocabulary contained in the source code is considered. In our work, we further develop the approach by Atzberger *et al.* , who utilized LLDA to extract the concept-specific vocabulary in comments and identifier names within source code [7]. This allows the determination of expertise in higher-level concepts and traceability at the commit level.

III. Approach

Our approach examines developers' commit history for their expertise with programming languages, libraries, and higher-level concepts. An overview of our data processing pipeline is shown in Figure 1. Using the GitHub API, we can collect the changes made by a developer. For programming languages, libraries, or higher-level concepts, words or, more generally, regular expressions are defined to represent their capabilities. In order to capture the words that indicate expertise in a higher-level concept, we rely on LLDA, trained on a corpus of representative projects taken from GitHub. A score is then determined based on the frequency of the expressions used, reflecting expertise in the area, similar to existing

[5]Screencast available at Youtube: https://youtu.be/1hB_QGM1eRg
[6]https://stackoverflow.com/

approaches [8], [10]. The results of our approach are shown on a web frontend, which further provides links to the actual commits of a developer.

A. *Vocabulary Associated to Skills*

Libraries and programming languages have a defined set of keywords. Programming languages are also characterized by individual idioms, which can be localized as regular expressions in the source code. Examples of idioms of the programming language Python would be, for instance:

1) **List Comprehension** Concise syntax for creating lists from other iterables or sequences. Regular expression: "\[.* for .*\]".
2) **Dict Comprehension** Concise syntax for creating dictionary objects from other iterables or sequences. Regular expression: "\{.* for .*\}"
3) **Slicing** Syntax for specifying a range of a sequence. Regular expression: "\[[a–zA–Z0–9_]*:[a–zA–Z0–9_]*\]"

For the used programming languages and libraries, the frequencies of the use of keywords and idioms are parsed from the commit history of a selected developer.

Unlike programming languages or libraries, it is initially unclear which words in the source code are suitable as indicators of expertise for higher-level concepts. In practice, however, it can be observed that conclusions about the underlying domain of a source code snippet can be drawn from the comments and the choice of identifiers. In numerous applications, it has been shown that topic models, especially LDA [11] and its variants, are suitable for analyzing natural language in source code [12], especially for analyzing developer expertise [7], [13], [14]. Given a description of a set of documents as a term-document matrix, LDA extracts the latent topics as distributions over the vocabulary and simultaneously describes the individual documents as distributions over the vocabulary [11]. Often, the most likely words of a topic can be used to guess its underlying concept. LDA is further able to handle synonyms. However, the topics must be labeled manually, which is a non-trivial endeavor.

Labeled LDA, a variant of LDA, overcomes this issue by analyzing documents with associated labels [15]. Each label is associated with precisely one topic; a document can only have an expression in this topic if it is marked with the corresponding label. Thus, we can extract the concept-specific vocabulary by applying LLDA to a corpus of source code documents with labels corresponding to our desired higher-level concepts. An approach to creating a corpus of representative source code projects was proposed by Atzberger *et al.* [7]. For each concept $c_1, ..., c_n$, all projects on GitHub marked with the respective GitHub topic are outputted and sorted by the number of likes in descending order. In addition, the number of open issues is intended to ensure that they are actual software projects. Projects are added to the corpus for each concept until the vocabulary stagnates, starting with the most popular project. The documents in the corpus then exist as a union of all source code files within the project. Table I shows the ten first representatives for the higher-level concepts *Machine Learning*, *Mobile*, and *Cryptocurrency*.

TABLE I
FIRST TEN PROJECTS FROM *GitHub* THAT ARE MARKED AS REPRESENTATIVE PROJECTS FOR THE CONCEPTS *Machine Learning*, *Mobile*, AND *Cryptocurrency*.

Machine Learning	Mobile	Cryptocurrency
tensorflow	flutter	bitcoin
transformers	ionic-framework	ccxt
pytorch	awesome-flutter	freqtrade
keras	fastlane	OpenBBTerminal
scikit-learn	awesome-react-native	lbry-sdk
tesseract	swiper	monero
face_recognition	vant	xmrig
TensorFlow-Examples	weex	blockchain
faceswap	framework7	lnd
julia	expo	tendermint

TABLE II
TOP 10 WORDS FOR THE HIGHER-LEVEL CONCEPTS *Machine Learning*, *Mobile*, AND *Cryptocurrency*. THE TOPICS ARISE FROM APPLYING LLDA ON THE AUTOMATICALLY GENERATED CORPUS OF *GitHub* PROJECTS.

Machine Learning	Mobile	Cryptocurrency
th	android	fe
tensor	view	order
self	name	symbol
cuda	com	crypto
input	get	binance
model	param	price
license	conv	trade
output	activity	wallet
layer	fpga	exchange
size	wishlist	block

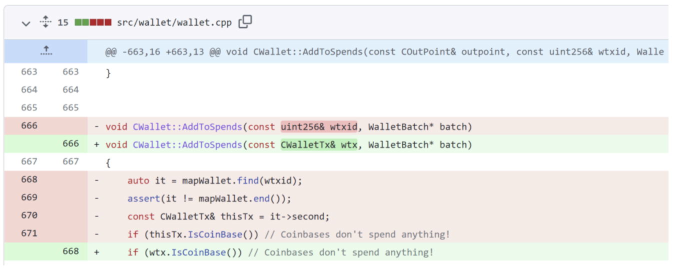
Before applying a topic model, we need to undertake the documents several preprocessing steps to remove words that carry no semantic meaning. We follow the preprocessing presented in [7]. After preprocessing the corpus and storing it as a term-document-matrix, we train an LLDA model, where each document is assigned with its higher-level concept and an additional topic that is used to capture the words relevant over all documents. To increase the interpretability of the topics, we apply the weighting scheme proposed by Sievert and Shirley [16]. Table II shows the top ten words for the higher-level concepts *Machine Learning*, *Mobile*, and *Cryptocurrency*. Now given the commit history of a developer, we apply the same preprocessing steps as before and count the frequencies of the top 30 words representing a topic.

B. *Scoring Function*

Let c denote a programming language, a library, or a higher-level concept with its indicating vocabulary $\mathcal{V}_c$. Our computation of the score relies on a set of developers $\mathcal{D}$ that is used to compute the relevance of a term for a concept. The relevance $r_{w,c}$ of a word $w \in \mathcal{V}_c$ for a concept c is given by

$$r_{w,c} = \frac{1}{N_{w,\mathcal{D}}} \sum_{w' \in \mathcal{V}_c} N_{w',\mathcal{D}}, \qquad (1)$$

where $N_{w,\mathcal{D}}$ denotes the overall count of term w in the commit histories of all developers from $\mathcal{D}$, i.e., words that occur less



```
15 src/wallet/wallet.cpp
@@ -663,16 +663,13 @@ void CWallet::AddToSpends(const COutPoint& outpoint, const uint256& wtxid, Walle
  }


- void CWallet::AddToSpends(const uint256& wtxid, WalletBatch* batch)
+ void CWallet::AddToSpends(const CWalletTx& wtx, WalletBatch* batch)
  {
-     auto it = mapWallet.find(wtxid);
-     assert(it != mapWallet.end());
-     const CWalletTx& thisTx = it->second;
-     if (thisTx.IsCoinBase()) // Coinbases don't spend anything!
+     if (wtx.IsCoinBase()) // Coinbases don't spend anything!
```

Fig. 2. Extract from a commit of the *Bitcoin* project, whose vocabulary has a large relative amount of keywords related to the concept *Cryptocurrency*.

frequent have a higher relevance for c. For a term $w \in \mathcal{V}_c$, let $N_{w,d}$ denote the frequency of word w within the commit history of developer d. Let $N_{\mathcal{V}_c,d}$ denote the overall count of word usage of terms from $\mathcal{V}_c$ within the commit history of developer d, i.e.,

$$N_{\mathcal{V}_c,d} = \sum_{w' \in \mathcal{V}_c} N_{w',d}. \tag{2}$$

The score of developer d in c is then determined by

$$s_{d,c} = \sum_{w \in \mathcal{V}_c} \frac{r_{w,c} \cdot N_{w,d}}{N_{\mathcal{V}_c,d}}. \tag{3}$$

The score increases when the author uses more words from the $\mathcal{V}_c$, especially rare words, as discussed in [17].

C. Exploring Results

The results for a developer are represented in a dashboard by two types of diagrams. The individual characteristics of the programming languages, libraries, and higher-level concepts considered are summarized in a radar chart. In addition, the achieved scores are presented comparatively with the values of the benchmark $\mathcal{D}$ in a box plot for each category.

When requesting the details for an applicant, in addition to a summary of their scores, specific commits that serve as concrete examples for the usage of the specific concept are of particular interest to the reviewer. Our system allows us to display such commits for a selected higher-level concept. For this purpose, the commits of a developer are outputted sorted by the relative number of terms from $\mathcal{V}_c$. Figure 2 shows an example of a commit from the *Bitcoin* project[7], which is recommended as an example for the use of the concept *Cryptocurrency*.

IV. Early Results

Our first experiments investigate to what extent expertise in higher-level concepts can be detected by our method. So far, we have refrained from a detailed evaluation of expertise in programming languages and libraries since their investigations do not rely on machine learning methods and are purely rule-based. Specifically, we look at the concepts of *Data visualization*, *Machine learning*, *Front end*, *Server*, *Database*,

[7]https://github.com/bitcoin/bitcoin/commit/174b821e64d61d7cd9466c7b73f71b2eb8508f92

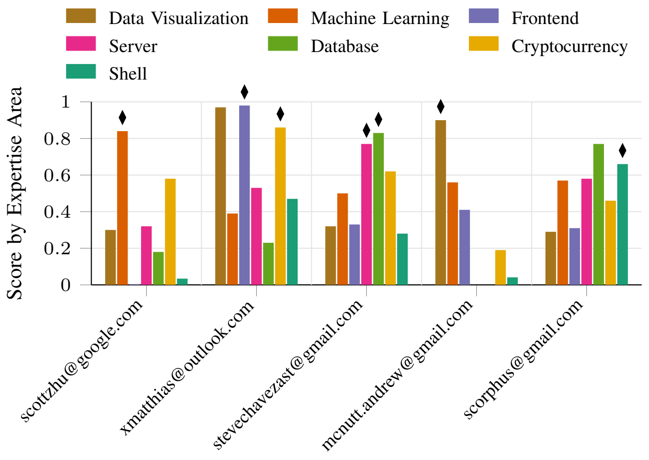


Fig. 3. Bar chart showing the achieved percentiles for the five selected GitHub users in comparison to the developers from the benchmark. The ⧫ indicates the expected field of expertise for each user.

Cryptocurrency, and *Shell*. These are representatives of the curated topic list from GitHub[8]. For this case study, we look at a selection of active GitHub developers. We selected some of the most frequent contributors to projects tagged with at least one of the concepts we consider in our approach. We then calculated their concept score percentiles based on commits to projects they most recently contributed. We consider the following GitHub users, with their expected field of expertise highlighted:

1) **scottzhu@google.com** One of the recent main contributors to the *keras*[9] machine learning library repository and software engineer in **machine learning** at Google, according to his GitHub profile.
2) **xmatthias@outlook.com** One of the most active authors of the *freqtrade*[10] **cryptocurrency** repository and its UI **frontend** repository *frequi*[11].
3) **stevechavezast@gmail.com** One of the most active developers of the repository *postgrest*[12], an http **server** for postgres **databases**.
4) **mcnutt.andrew@gmail.com** One of the top authors of the **data visualization** library *react-vis*[13].
5) **scorphus@gmail.com** A top contributor to the **shell** command correction tool *thefuck*[14].

Figure 3 shows the resulting score percentiles for the developers. Our benchmark $\mathcal{D}$ comprises 1448 active GitHub users. For each concept, we make sure to have at least 200 developers, where each has made at least 200 commits.

For the first four developers, the expected fields of expertise are among the highest percentiles of that developer. An exception is the user scorphus@gmail.com, who scores highest in

[8]https://github.com/topics
[9]https://github.com/keras-team/keras
[10]https://github.com/freqtrade/freqtrade
[11]https://github.com/freqtrade/frequi
[12]https://github.com/PostgREST/postgrest
[13]https://github.com/uber/react-vis
[14]https://github.com/nvbn/thefuck

the *database* concept, and only second-highest in *shell*, despite being an active contributor in this field. The reason for that is that the indicator words for the concepts *Database* and *Shell* overlap in many parts. Also the two concepts *Data Visualization* and *Front end* share large parts of their vocabularies, which implies a high score for xmatthias@outlook.com in the *Data Visualization* concept. To summarize, our results show that our approach is able to detect expertise in more than one higher-level concept.

Our primary concern is whether, with a more significant number of concepts, the vocabulary is suitable for identifying the respective expertise. Furthermore, it is not clear a priori whether, for particular concepts, suitable examples are available on GitHub from which the specific vocabulary can be extracted.

V. Conclusions

Mining Developer Expertise for GitHub users is of great interest to recruiters. We proposed a data processing and visualization approach based on the actual commit activities. Besides a rule-based technique for capturing expertise in programming languages and libraries, we apply LLDA to locate expertise in higher-level concepts. Our proposed scoring function is built to capture the frequency and variety of vocabulary associated with each category. Our prototype provides an interactive dashboard that compares the achieved scores within a benchmark and allows traceability to the actual commits. Our first results indicate that our approach can detect higher-level expertise in commit histories. It also suggests that our score function is well suited for describing the skills in programming languages, libraries, and higher-level concepts.

One limitation is that we do not match several GitHub profiles that belong to the same developer. Furthermore, our score function relies on a set of developers, and we need to evaluate the effect on the score function when modifying the benchmark. Another possible direction for future work is to improve the quality of indicator words by applying different selection mechanisms, combining hyperparameter tuning in the LLDA model, and weighting the resulting topics. GitHub also provides the *best match* mechanism for detecting exemplary projects for a given GitHub topic, which might result in a better-suited training corpus for the LLDA model and influence the quality of the indicating words for a higher-level concept. However, a more extensive evaluation is the most crucial part of future work. We plan to conduct a user study with possible recruiters to evaluate the effectiveness of our system. Furthermore, we plan to apply the analysis and visualization approach to other knowledge management tasks within a company.

Acknowledgements

We want to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. This work is part of the "Software-DNA" project, which is funded by the European Regional Development Fund (ERDF or EFRE in German) and the State of Brandenburg (ILB). This work is part of the KMU project "KnowhowAnalyzer" (Förderkennzeichen 01IS20088B), which is funded by the German Ministry for Education and Research (Bundesministerium für Bildung und Forschung).

References

[1] Sandeep Kaur Kuttal, Xiaofan Chen, Zhendong Wang, Sogol Balali, and Anita Sarma. Visual resume: Exploring developers' online contributions for hiring. *Information and Software Technology*, 138:106633, 2021.

[2] Gillian J. Greene and Bernd Fischer. CVExplorer: Identifying candidate developers by mining and exploring their open source contributions. In *Proc. 31st International Conf. on Automated Software Engineering*, ASE '16, pages 804–809. IEEE/ACM, 2016.

[3] Claudia Hauff and Georgios Gousios. Matching GitHub developer profiles to job advertisements. In *Proc. 12th Working Conf. on Mining Software Repositories*, MSR '15, pages 362–366. IEEE/ACM, 2015.

[4] Stratos Kourtzanidis, Alexander Chatzigeorgiou, and Apostolos Ampatzoglou. RepoSkillMiner: Identifying software expertise from GitHub repositories using natural language processing. In *Proc. 35th International Conf. on Automated Software Engineering*, ASE '16, pages 1353–1357. IEEE/ACM, 2020.

[5] Rohit Saxena and Niranjan Pedanekar. I know what you coded last summer: Mining candidate expertise from GitHub repositories. In *Companion of the 2017 Conf. on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 299–302. ACM, 2017.

[6] David Ma, David Schuler, Thomas Zimmermann, and Jonathan Sillito. Expert recommendation with usage expertise. In *Proc. 25th International Conf. on Software Maintenance*, ICSM '09, pages 535–538. IEEE, 2009.

[7] Daniel Atzberger, Tim Cech, Adrian Jobst, Willy Scheibel, Daniel Limberger, Matthias Trapp, and Jürgen Döllner. Visualization of knowledge distribution across development teams using 2.5D semantic software maps. In *Proc. 17th International Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications*, IVAPP '22, pages 210–217. INSTICC, SciTePress, 2022.

[8] Cédric Teyton, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc. Find your library experts. In *Proc. 20th Working Conf. on Reverse Engineering*, WCRE '13, pages 202–211. IEEE, 2013.

[9] Cédric Teyton, Marc Palyart, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc. Automatic extraction of developer expertise. In *Proc. 18th International Conf. on Evaluation and Assessment in Software Engineering*, EASE '14, pages 1–10. ACM, 2014.

[10] João Eduardo Montandon, Luciana Lourdes Silva, and Marco Tulio Valente. Identifying experts in software libraries and frameworks among GitHub users. In *Proc. 16th International Conf. on Mining Software Repositories*, MSR '19, pages 276–287. IEEE/ACM, 2019.

[11] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] Tse-Hsun Chen, Stephen W. Thomas, and Ahmed E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5):1843–1919, 2016.

[13] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. In *Proc. 4th International Workshop on Mining Software Repositories*, MSR '07, pages 30:1–4. IEEE, 2007.

[14] Maria Papoutsoglou, Johannes Wachs, and Georgia M. Kapitsaki. Mining DEV for social and technical insights about software development. In *Proc. 18th International Conf. on Mining Software Repositories*, MSR '21, pages 415–419. IEEE/ACM, 2021.

[15] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In *Proc. 2009 Conf. on Empirical Methods in Natural Language Processing*, EMNLP '09, pages 248–256. ACL, 2009.

[16] Carson Sievert and Kenneth Shirley. LDAvis: A method for visualizing and interpreting topics. In *Proc. Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70. ACL, 2014.

[17] Arghavan Moradi Dakhel, Michel C. Desmarais, and Foutse Khomh. Assessing developer expertise from the statistical distribution of programming syntax patterns. In *Proc. 25th Conf. on Evaluation and Assessment in Software Engineering*, EASE '21, pages 90–99. ACM, 2021.